UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/858,241	05/15/2001	Carlos L. Thompson	10001214-1	3049

7590 01/21/2005

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, CO 80527-2400

EXAMINER	
INGBERG, TODD D	
ART UNIT	**PAPER NUMBER**
2124	

DATE MAILED: 01/21/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application N .	Applicant(s)	
	09/858,241	THOMPSON ET AL.	
	Examiner	Art Unit	
	Todd Ingberg	2124	

-- The MAILING DATE f this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *31 August 2004*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-30* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1,6-8,11,13-16,21-23 and 28-30* is/are rejected.

7)☐ Claim(s) *2-5,9,10,12,-24-27,17-20* is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *5/15/2001* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

Claims 1 – 15 have been examined.

Claims 16 – 30 have been added.

Knowledge of the Artisan

1. One of ordinary skill in the art must understand the basics of compiler theory.

The basic college text book "Compilers Tools and Techniques", by Aho et al. from September 1985 is considered grossly old and well known. This text book is not used in the rejection.

Claim Rejections - 35 USC § 101

2. 35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

Claim 30 is rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter. One way to overcome the rejection is to amend the claim as follows.

Claim 30
An apparatus **stored on a computer readable medium and executing on a computer** for verifying at runtime an invariant property of a data structure of a computer program, comprising: means for determining an invariant property of a data structure from a source code specification of the data structure and an associated specification of the invariant property in the source code, wherein the specification of the invariant property defines the invariant property without checking whether a variable used with the data structure is consistent with the invariant property; means for generating from the specification of the invariant property a first executable
code segment that determines whether a value of a variable used with the data structure is consistent with the invariant property;
means for, determining during execution of the. first executable code segment whether the value of the variable used with the data structure is consistent with the invariant property; and means for performing a programmed action in response to the value of the variable being inconsistent with the invariant property.

Claim Rejections - 35 USC § 102

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

3. Claims 1, 6, 7,-8, 11, 13 – 14, 22-23, 28-29, 15-16, 21 and 30 are rejected under 35 U.S.C. 102(b) as being anticipated by **Asuru** in the ACM publication, "Optimization of Array Subscript Range Checks" from 1992.

Claim 1

Asuru anticipates a computer-implemented method for verifying at runtime an invariant property of a data structure of a computer program, comprising: automatically generating a first code segment that verifies a runtime value of the data structure is consistent with the invariant property in response to an annotation of the data structure that defines the invariant property of the data structure; comparing the runtime value of the data structure with the invariant property during execution of the program via execution of the first code segment; and performing a programmed action if the runtime value is inconsistent with the invariant property.

Examiner's Rejection

Asuru anticipates the verifying of the loop invariant data structure at runtime(page 113 Range Check - Introduction) . The runtime range values are determined and replace the operands of the range (page 110 first paragraph). The upper and lower bounds the invariant property of the data structure as per above. The Range is used at runtime to ensure the loop invariant is not out of bounds with the subscripts (range) as per above.

Claim 6

The method of claim 1, wherein the invariant property is a range of data values and further comprising the step of verifying that the runtime value of the data structure is within the range of data values.

Examiner's Rejection

As per claim 1.

Claim 7

The method of claim 1, further comprising communicating the invariant property from a compiler to a code generator.

Examiner's Rejection

As per claim 1 - inherent step.

Claim 8

The method of claim 7, further comprising storing the invariant property in a symbol table.

Examiner's Rejection

As per claim 1 – Inherent the invariant is a loop control variable – all variables are inherently stored in the symbol table.

Claim 11

The method of claim 8, wherein the invariant property is a range of instruction addresses and further comprising verifying that the runtime value of the data structure is within the range of instruction addresses specified in source code of the computer program.

Examiner's Rejection

As per claim 1 - inherent step – the range is stored in a variable. All entries in the symbol table have an address.

Claim 13

The method of claim 8, wherein the invariant property is a range of data values and further comprising the step of verifying that the runtime value of the data structure is within the range of data values.

Examiner's Rejection

As per claim 1 - inherent step – the range is stored in a variable. All entries in the symbol table have an address.

Claim 14

The method of claim 8, further comprising storing in the symbol table one or more code addresses associated with one or more updates to the data structure.

Examiner's Rejection

As per claim 1 - inherent step – the range is stored in a variable. All entries in the symbol table have an address. Symbol Tables inherently support updates and more than one symbol being defined.

Claim 22

The method of claim 1, further comprising communicating the invariant property from a compiler to a code generator.

Examiner's Rejection

As per claim 1 - inherent step – the range is stored in a variable. All entries in the symbol table have an address.

Claim 23

The method of claim 22, further comprising storing the invariant property in a symbol table.

Examiner's Rejection

As per claim 1 - inherent step – the range is stored in a variable. All entries in the symbol table have an address.

Claim 28

The method of claim 23, wherein the invariant property is a range of data values and the step of determining whether the value of the variable is consistent with the invariant property includes determining whether the value of the variable used with the data structure is within the range of data values.

Examiner's Rejection

As per claim 1 - inherent step – the range is stored in a variable. All entries in the symbol table have an address.

Claim 29

The method of claim 23, further comprising storing in the symbol table one or more instruction addresses at which respective updates are made to the data structure.

Examiner's Rejection

As per claim 1 - inherent step – the range is stored in a variable. All entries in the symbol table have an address and the symbol table hold more than one symbol.

Claim 16

A computer-implemented method for verifying at runtime an invariant property of a data structure of a computer program, comprising:

determining an invariant property of a data structure from a source code specification of the data structure and an associated specification of the invariant property in the source code, wherein the specification of the invariant property defines the invariant property without checking whether a variable used with the data structure is consistent with the invariant property;

generating from the specification of the invariant property a first executable code segment that determines whether a value of a variable used with the data structure is consistent with the invariant property; determining during execution of the first executable code segment whether the value of the variable used with the data structure is consistent with the invariant property; and performing a programmed action in response to the value of the variable being inconsistent with the invariant property.

Examiner's Rejection

As per claim 1

Claim 21

The method of claim 16, wherein the invariant property is a range of data values and the step of determining whether the value of the variable is consistent with the invariant property includes determining whether the value of the variable used with the data structure is within the range of data values.

Examiner's Rejection

As per claim 1

Claim 30

An apparatus for verifying at runtime an invariant property of a data structure of a computer program, comprising: means for determining an invariant property of a data structure from a source code specification of the data structure and an associated specification of the invariant property in the source code, wherein the specification of the invariant property defines the invariant property without checking whether a variable used with the data structure is consistent with the invariant property; means for generating from the specification of the invariant property a first executable code segment that determines whether a value of a variable used with the data structure is consistent with the invariant property; means for, determining during execution of the. first executable code segment whether the value of the variable used with the data structure is consistent with the invariant property; and means for performing a programmed action in response to the value of the variable being inconsistent with the invariant property.

Examiner's Rejection

As per claim 1

4. Claim15 is rejected under 35 U.S.C. 102(b) as anticipated by or, in the alternative, under 35 U.S.C. 103(a) as obvious over Runtime error for subscript out of range errors as taught by **Sirer** in the article Writing an Operating System with Modula-3 November 3, 1995.

Claim 15

An apparatus for verifying at runtime an invariant property of a data structure of a computer program, comprising: means for automatically generating a first code segment that verities a runtime value of the data structure is consistent with the invariant property in response to an annotation of the data structure that defines the invariant property of the data structure; means for comparing the runtime value of the data structure with the invariant property during execution of the program via execution of the first code segment; and means for performing a programmed action if the runtime value is inconsistent with the invariant property.

Examiner's Rejection

Asuru anticipates the verifying of the loop invariant data structure at runtime(page 113 Range Check - Introduction) . The runtime range values are determined and replace the operands of the range (page 110 first paragraph). The upper and lower bounds the invariant property of the data structure as per above. The Range is used at runtime to ensure the loop invariant is not out of bounds with the subscripts (range) as per above. One of ordinary skill in the art should know that run time error checking is inherent and that subscript out of range errors generate an exception handler that invoke the operating.

In the event, the runtime environment error checking is challenged then the Examiner provides the MODULA Operating System reference by Sirer from November 3, 1995 Section 5 which proves providing for run time error checking such as bounds checking error (page 3) is grossly old and well known. Therefore, it would be obvious to one of very ordinary skill in the art to combine Asuru and Sirer, because handling error correctly ensures programs run properly.

Allowable Subject Matter

5. Claims 2-5, 9, 10, 12, 24-27 and 17-20 objected to as being dependent upon a rejected base claim, but would be allowable if rewritten in independent form including all of the limitations of the base claim and any intervening claims.

The objected claims specify a range of address. The Geva reference mentions the upper bound of the invariant can be the value which is supported by the prior art. Geva also mentions the invariant can be an address (memory location) column 4 lines 5 – 10, but this is a specific reference to the pointer of the memory location containing a data value not a range. The loop unrolling reference also covers the formulating of a loop invariant. And continues to mention that the calculated invariant may not be able to be used at runtime. this is because with Do-While loops the number of iterations is not known until runtime. Unlike For-Loops where the iterations

is fixed during runtime or compile time. Also, Geva teaches loop unrolling which can involve code motion which means moving code outside a loop construct during loop unrolling which means the invariant calculated during compile time is not used. Search of prior art did not result in meeting the limitations of claims if rolled into the independent claims.

Prior art of record failed to teach the limitations in combination with the independent claims singularly or in combination. Examiner urges Applicant to use caution toward 112 second paragraph to avoid antecedent basis issues.

Response to Arguments

6. Applicant's arguments filed August 31, 2004 have been fully considered but they are not fully persuasive.

Applicant's Argument

"Claims 1-15 stand rejected under 35 USC § 102(b) as being anticipated by the publication, "Optimization of Array Subscript Range Checks" by Asuru (hereinafter Asuru). The rejection is respectfully traversed because the Office Action does not show that all the limitations are taught by Asuru.

The invention of claim 1 is a method for verifying at runtime an invariant property of a data structure of a computer program. The limitations include automatically generating a first code segment that verifies a runtime value of the data structure is consistent with the invariant property in response to an annotation of the data structure that defines the invariant property of the data structure; comparing the runtime value of the data structure with the invariant property during execution of the program via execution of the first code segment; and performing a programmed action if the runtime value is inconsistent with the invariant property. The Office Action mistakenly interprets Asuru as teaching these limitations.

For example, the limitations specifically call out generating code that verifies the runtime value of the data structure relative to the invariant property and in response to the annotation of the data structure that defines the invariant property. Asuru does not teach generating code, nor does Asuru teach that the code is generated from the annotation that defines the invariant property."

Examiner's Response

Applicant appears to be arguing that since Asuru teaches more than the certain claims claimed invention the claimed invention is patentable. This argument is not persuasive.

The optimization of runtime invariants still teaches the generation of invariants. The invariants must be generated in order to test them. Asuru goes beyond the generation of invariants, additionally teaches optimize.

After having read the Applicant's arguments it appears Applicant is arguing the basic terms and the meaning of an invariant and how an invariant applies to a loop as taught by the Asuru reference.

In FAOM the Examiner stated: "One of ordinary skill in the art must understand the basics of compiler theory. The basic college text book "Compilers Tools and Techniques", by Aho et al. from September 1985 is considered grossly old and well known". Applicant's arguments for independent claims are toward inherent aspects of the art as outlined by the following:

I. Basic Compiler Theory
   a. lexical analysis
   b. symbols
   c. instructions & symbols with inherent addressing
II. What is a loop and how lexical analysis translates it
III. What is mean by bounds of a loop
IV. What is an invariant
V. What is a Loop Invariant
VI. What is a runtime check of a loop invariant
VII. How does the term "Array Subscript Range Checks" relate to loop invariant
VIII. Why Optimization of loop invariants is not the opposite of their invention.
IX. How Inherent Principles Relate to the Asuru Reference

These grossly old and well known and inherent concepts should be well known to one of ordinary skill in the art well before embarking on an invention in the compiler field with runtime checking of invariants.

I. Basic Compiler Theory

Inherent aspects of compiler theory have gone ignored. Asuru tests values established from compilation time with runtime information. The presence of the bounds of a loop are evidence of the by product of a compiler.

a. Lexical analysis

Lexical analysis is how source code is translated. Aho on page 5 provides a simple view of an assignment statement. Note the identifiers and values assigned.

b. Symbols

Symbol-Table Management is described by Aho on page 11 in the "Introduction to Compiling " section as an essential function of a compiler. The symbol table is defined as "a data structure containing a record for each identifier, with fields for the attributes of the identifier."

Furthermore, Aho connects the symbol table to lexical analysis in the section. Which supports the figure on page 10.

The Aho reference provides many mentions to the symbol table. Suggested reading is on c. Instructions & symbols with inherent addressing

Amended claims appear that Applicant believes they invented addressing. Prior art proves a loop invariant is stored in a data structure provided by the compiler inherently. Also, all *idenitiers* and instructions inherently have an address. See Aho, pages 13-14 and 446-472.

II. What is a loop and how lexical analysis translates it

On pages 596 – 598 Aho mentioned loop optimization. If one does not fixate on the term optimization and looks at figure 10.9 on page 597 and notices the (a) Before (As in non optimized loop) the figure shows a loop structure. Building on the inherent principles of compiler theory the Examiner has gone to the level of less than ordinary skill in the art to point out, one can see program statements. Variables like i, j, t and v are on the left side of the assignment statement. These are identifiers and place in a symbol table. The sections above should be reread if this is not clear.

III. What is mean by bounds of a loop

Pages 638 – 639 for Detection of Loop-Invariant Computation has already been mentioned. In addition to this section an easier read is the Muchnick reference pages 454 – 457. When one ignores the fact that it is teaching optimization and only focuses on the inherent aspects of loops and invariants one can see the same constructs as found in Aho and Asuru.

IV. What is an invariant

Applicant's provided a definition for the term ***"invariant"*** from a foreign website and after the original filing date. This definition does not distinguish the invention from the use of loop invariant checking as taught by Asuru. The definition provided is ", an invariant is "a rule, such as the ordering of an ordered list or heap, that applies throughout the life of a data structure or procedure". The Examiner meets this definition with the Asuru reference by providing loop invariant checking in the Asuru reference. The fact that Asuru goes beyond the claimed invention and optimizes loop invariant checking is not grounds for patentablility. To emphasis that loop invariant checking is invariant checking the Morgan reference on page 204 defines the term "Loop Invariant" as A temporary T is a loop invariant in the loop L if it is either not computed in the loop or its operands are loop invariants. This meets the Applicant's definition by stating the loop control by calculating a valid range and testing the loop control variable meets both definitions. Furthermore, the inherent Detection of Loop-Invariant Computations is on pages 638 – 639 Aho the inherent steps include; detecting, computing and marking loop invariant.

V. What is a Loop Invariant

As mentioned above the range of values a loop control variable can assume is a range the lower value is a lower bound and higher value is a upper bound. Testing the loop control variable to ensure it is within this range is an invariant test. This is a loop invariant. The value of the loop control variable (subscript) must be within this range.

VI. What is a runtime check of a loop invariant

A runtime check of a loop invariant is a test to see if the value of the loop control variable (subscript) is within the allowed range (range of loop counter not address range).

VII. How does the term "Array Subscript Range Checks" relate to loop invariant

The value of the loop control variable (subscript) if outside the range is considered a bounds violation. Testing the invariant is to test if the subscript is with in the allowed data range at runtime.

VIII. Why Optimization of loop invariants is not the opposite of their invention.

As stated above the optimizing of loop invariants is not an elimination of loop invariant testing. The safety is not violated. Redundant tests are eliminated in one form of optimization. It is very critical to note this is not opposite of your invention.

IX. How Inherent Principles Relate to the Asuru Reference

The Asuru reference explicitly teaches range checking as per claim 1. Range checking is a form of bounds checking which is also explicitly mentioned. The range is checked to see if the range of the loop counter is in range. The range values (bounds) are annotations. The generation of code to enforce range checking is inherent in compiler theory. Examiner in FAOM held the teaching of Aho et al in the September 19, 1985 college text book, "Principles, Compilers, Tools and Techniques", shows the inherent "Detection of Loop-Invariant Computation" on pages 638-639. The invariant for a loop is not only detected but also marked. the Asuru reference optimizes invariants. The fact that Asuru optimizes invariants for loops is evidence that the structures were present to be optimized. Note the optimization does not violate the safety constraints by performing range check analysis (Asuru page 112 3. Range Check Optimization. In other words, the invariant test is still present after the optimization. The Asuru reference teaches more than the claimed invention of certain independent claims.

Applicant's Argument

"Asuru specifically teaches eliminating redundant range check instructions (page 109, Abstract; page 109, last line), which those skilled in the art would recognize to be opposite of automatically generating the code that verifies the invariant property. Furthermore, assuming for discussion purposes only that eliminating redundant range checks could be construed as automatically generating code, Asuru's elimination of the redundant checks is not in response to the annotation of a data structure that defines the invariant property. Rather Asuru's method includes the three steps: "(1) identify and eliminate inner loop guards, (2) replace subscript operands of range checks with expressions which yield either the least value (lower bound check) or the greatest value (upper bound check) of the subscript operands, and (3) perform range check optimization." (page 110). It is respectfully submitted that Asuru's range checks. verify run-time compliance with an invariant property, but are not an-annotation that defines the invariant property."

Examiner's Response

Applicant's statement that "... eliminating redundant range check instructions is opposite of automatically generating the code that verifies the invariant property", is false. First, the detection and marking of loop invariants is inherent in compiler theory. Second, Asuru is only eliminating redundant checks. This is called optimization. The fact that Asuru performs invariant optimization for a loop is evidence the loop invariants are present. The steps of loop invariant detection, computation and marking are inherent in compiler theory (As per above).

Applicant's Argument

"The rejections of claims 2-15 are similarly in error. That is, in Asuru the range checking code is present in the program and Asuru seeks to eliminate redundant range checking code. There is no apparent teaching in Asuru that the range checking code is automatically generated from an annotation of a data structure, and the cited teaching does not appear to suggest that there is any annotation from which the code is generated."

Examiner's Response

As stated above the data values which make the range (upper and lower bounds) are annotations. A range test is not possible without knowing what the correct range (upper and lower bounds). The range is detected, calculated and marked by the compiler (Aho, pages 638-639).

Applicant's Argument

"The Office Action also alleges that certain limitations are inherent. However, in alleging inherency the Office Action ignores limitations in the claims: For example, in claim 4 the invariant property is a range of instruction addresses, and the method verifies that the runtime value of the data structure is within the range of instruction addresses specified in source code of the computer program. The Office Action explains, "As per claim 1 - inherent step - the range is stored in a variable. All entries in the symbol table have an address." It is respectfully submitted that the claimed range is of instruction addresses, not just any data address, and instruction addresses are not believed to be inherently specified in Asuru's source code. Therefore, the Office Action does not show that the limitations are inherent in Asuru."

Examiner's Response

By definition a range check is a test of value of the loop control variable as it relates to its upper and lower bounds. This limitation is present and was met by the rejection. A loop is an instruction. The range the loop control variable can have relates to the range the test to ensure the loop operation is within range is called a range check (bounds test). This invariant test meets the claimed invention.

Applicant's argument that the claims with "range of instruction addresses" is persuasive.

Applicant's Argument

"The rejections of the claims based on inherency are also improper because the Office Action does not provide a basis in fact and/or technical reasoning to reasonably support the determination that the allegedly inherent characteristic necessarily flows from the teachings of the applied prior art (MPEP 2112). A citation to supporting art is requested if the allegation of inherency is maintained so that the substance of the alleged inherency may be reasonably reviewed."

Examiner's Response

The challenge to inherency has been met in the response above. The rejection was written to one of ordinary skill in the art at the time of invention.

Applicant's Argument

"As to claim 8, the limitations include storing the invariant property in a symbol table, and the Office Action mistakenly alleges that since variables are stored in a symbol table, an invariant property would be stored in a symbol table. This reasoning, to the extent it is understood, apparently equates a variable to an invariant property. However, those skilled in the art will clearly recognize that a variable is not the same as an invariant property.

From the Free On-Line Dictionary Of Computing (FOLDOC) at http://foldoc.doc.ic.ac.uk/foldoc/index.fktrnl, an invariant is "a rule, such as the ordering of an ordered list or heap, that applies throughout the life of a data structure or procedure. Each change to the data structure must maintain the correctness of the invariant", and a variable is "a named memory location in which a program can store intermediate results and from which it can read it them.", Thus, identifying a variable in a symbol table does not suggest storing an invariant property in a symbol table."

Examiner's Response

Applicant's definition for the term "invariant" from a foreign website and after the original filing date, does not distinguish the invention from the use of loop invariant checking as taught by Asuru. In response, to Applicant's argument the Examiner will apply additional art to the new claims to emphasis the testing of a loop invariant is proper. The additional reference should also assist Applicant in understanding the inherent concepts of compiler theory they have amended directly into. The Morgan reference on page 204 defines the term "Loop Invariant" as A temporary T is a loop invariant in the loop L if it is either not computed in the loop or its operands are loop invariants. In other words, the address range or loop counter are invariants.

The Applicant's argument that "a variable in a symbol table does not suggest storing an invariant property in a symbol table", is false. As noted in the definition for a loop invariant from December 26, 1997 [Morgan], the loop operands can be loop invariants. The program construct: (Examiner made Example to emphasis basic concepts)

```
DO I From 1 to MAXVALUE
        statement
END LOOP
```

The constant 1 and MAXVALUE are both identifiers in the symbol table. This is a basic inherent concept in compiler theory see the college text book Aho the Introduction to Compilers on pages 10 - 11 and the figure on page 13 to understand the basics on symbol tables and how they relate to a line of code. The loop control variable I must be within the range of 1 to the value of MAXVALUE to be valid. Applicant's argument is not reflective of the inherent principles of compiler theory and the definition does not distance the claimed invention from the use of loop invariants.

Applicant's Argument

"The rejection of claim 15 under 35 USC §102(b) as being anticipated by the article entitled, "Writing an Operating System with Modula-3" by Sirer (hereinafter "Sirer"), or in the alternative under § 103(a) as being unpatentable over Sirer in view of Asuru, is respectfully traversed. Asuru does not teach the claim limitation as explained above. Furthermore, no teaching of Sirer is cited that shows generating invariant-checking code in response to the annotation of a data structure, and the alleged motivation for combining the references is conclusory. Therefore, the Office Action does not show anticipation by Sirer nor does the Office Action establish prima facie obviousness with the Asuru-Sirer combination."

Examiner's Response

Applicant's characterization of the use of Sirer is in clear error. The Examiner introduced the reference to illustrate on possible visible effect of violating a runtime invariant. A violation of a range check in the runtime environment is often referred to as a bounds error or a subscript out of range error. This is an example a programmer would recognize as a logical runtime error. The reference was added only to emphasis how common runtime detection of invariants checking is to one of ordinary skill in the art.

Examiner's Disposition

Applicant's arguments toward independent claims are tantamount to arguing that the Asuru reference teaches more than the claimed invention so the claimed invention is patentable. The Applicant has not acknowledged the presence of invariant checking in Asuru and has mischaracterized the reference as doing the opposite of the invention when it is optimizing. The invariant testing must be present in order to optimize. The term invariant as provided from a foreign website only strengthens the Examiners position. The Examiner's burden is met with the rejection of one form of invariant testing the range test of a loop. Applicant's argument's are not persuasive.

Applicant's argument that the claims with "range of instruction addresses" is persuasive. The range of a loop invariant is a data value range, not a "range of instruction addresses".

Conclusion

7. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

A. In First Action On Merit (FAOM) Examiner noted the inherent aspects of compiler theory are contained in the college text book by Aho. This reference is provided in an effort to accelerate prosecution.

8. Applicant's original claims were for the runtime environment which relied on the inherent features of compiler theory. New claims are directed toward the inherent features of a compiler. The new claims are for compile time. Restriction by Original Presentation was avoided by proper linkage of the original claims and new claims.

9. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Correspondence Information

10. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Todd Ingberg whose telephone number is (571) 272-3723. The examiner can normally be reached on during the work week..

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kakali Chaki can be reached on (571) 272-3719. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Todd Ingberg
Primary Examiner
Art Unit 2124

TI